Using SGI omplace for Pinning

Category: Process Pinning

Summary: The **`omplace`** wrapper script pins processes and threads for better performance. It works with SGI MPT, Intel MPI, and hybrid MPI/OpenMP applications.

SGI's **`omplace`** is a wrapper script for **`dplace`**. It provides an easier syntax than **`dplace`** for pinning processes and threads. **`omplace`** works with SGI MPT as well as with Intel MPI. In addition to pinning pure MPI or pure OpenMP applications, **`omplace`** can also be used for pinning hybrid MPI/OpenMP applications.

A few issues with **`omplace`** to keep in mind:

- **`dplace`** and **`omplace`** do not work with Intel compiler versions 10.1.015 and 10.1.017. Use the Intel compiler version 11.1 or later, instead
- To avoid interference between **`dplace/omplace`** and Intel's thread affinity interface, set the environment variable KMP_AFFINITY to **`disabled`** or set OMPLACE_AFFINITY_COMPAT to ON
- The **`omplace`** script is part of SGI's MPT, and is located under the **`/nasa/sgi/mpt/`*`mpt_version_number`*`/bin`** directory

Syntax

```
For OpenMP:
setenv OMP_NUM_THREADS nthreads
omplace [OPTIONS] program args...
or
omplace -nt nthreads [OPTIONS] program args...

For MPI:
mpiexec -np nranks omplace [OPTIONS] program args...

For MPI/OpenMP hybrid:
setenv OMP_NUM_THREADS nthreads
mpiexec -np nranks omplace [OPTIONS] program args...
or
mpiexec -np nranks omplace -nt nthreads [OPTIONS] program args...
```

Some useful **`omplace`** options are listed below:

-b *basecpu*
    Specifies the starting CPU number for the effective CPU list.
-c *cpulist*

Specifies the effective CPU list. This is a comma-separated list of CPUs or CPU ranges.

WARNING: For **`omplace`**, a blank space is required between **`-c`** and **`cpulist`**. Without the space, the job will fail. This is different from **`dplace`**.

-nt *nthreads*

Specifies the number of threads per MPI process. If this option is unspecified, it defaults to the value set for the OMP_NUM_THREADS environment variable. If OMP_NUM_THREADS is not set, then *nthreads* defaults to 1.

-v

Verbose option. Portions of the automatically generated placement file will be displayed.

-vv

Very verbose option. The automatically generated placement file will be displayed in its entirety.

For information about additional options, see **man omplace**.

Examples

For Pure OpenMP Codes Using the Intel OpenMP Library

Sample PBS script:

```
#PBS -lselect=1:ncpus=12:model=wes

module load comp-intel/11.1.072
setenv KMP_AFFINITY disabled

omplace -c 0,3,6,9 -vv ./a.out
```

Sample placement information for this script is given in the application's **`stout`** file:

```
omplace: placement file /tmp/omplace.file.21891
    firsttask cpu=0
    thread oncpu=0 cpu=3-9:3 noplace=1  exact
```

The above placement output may not be easy to understand. A better way to check the placement is to run the **`ps`** command on the running host while the job is still running:

```
ps -C a.out -L -opsr,comm,time,pid,ppid,lwp > placement.out
```

Sample output of **`placement.out`**

```
PSR COMMAND             TIME   PID  PPID   LWP
  0 openmp1         00:00:02 31918 31855 31918
 19 openmp1         00:00:00 31918 31855 31919
```

```
  3 openmp1         00:00:02 31918 31855 31920
  6 openmp1         00:00:02 31918 31855 31921
  9 openmp1         00:00:02 31918 31855 31922
```

Note that Intel OpenMP jobs use an extra thread that is unknown to the user, and does not need to be placed. In the above example, this extra thread is running on logical core number 19.

For Pure MPI Codes Using SGI MPT

Sample PBS script:

```
#PBS -l select=2:ncpus=12:mpiprocs=4:model=wes

module load comp-intel/11.1.072
module load mpi-sgi/mpt.2.04.10789

#Setting MPI_DSM_VERBOSE allows the placement information
#to be printed to the PBS stderr file

setenv MPI_DSM_VERBOSE

mpiexec -np 8 omplace -c 0,3,6,9 ./a.out
```

Sample placement information for this script is shown in the PBS **`stderr`** file:

```
MPI: DSM information
MPI: using dplace
grank   lrank   pinning  node name      cpuid
    0       0   yes      r144i3n12          0
    1       1   yes      r144i3n12          3
    2       2   yes      r144i3n12          6
    3       3   yes      r144i3n12          9
    4       0   yes      r145i2n3           0
    5       1   yes      r145i2n3           3
    6       2   yes      r145i2n3           6
    7       3   yes      r145i2n3           9
```

In this example, the four processes on each node are evenly distributed to the two sockets (CPUs 0 and 3 are on the first socket while CPUs 6 and 9 on the second socket) to minimize contention. If **`omplace`** had not been used, then placement would follow the rules of the environment variable **`OMP_DSM_DISTRIBUTE`**, and all four processes would have been placed on the first socket -- likely leading to more contention.

For MPI/OpenMP Hybrid Codes Using SGI MPT and Intel OpenMP

Proper placement is more critical for MPI/OpenMP hybrid codes than for pure MPI or pure OpenMP codes. The following example demonstrates the situation when no placement instruction is provided and the OpenMP threads for each MPI process are stepping on one another which likely would lead to very bad performance.

Sample PBS script without pinning:

```
#PBS -l select=2:ncpus=12:mpiprocs=4:model=wes

module load comp-intel/11.1.072
module load mpi-sgi/mpt.2.04.10789
setenv OMP_NUM_THREADS 2

mpiexec -np 8 ./a.out
```

There are two problems with the resulting placement shown in the example above. First, you can see that the first four MPI processes on each node are placed on four cores (0,1,2,3) of the same socket, which will likely lead to more contention compared to when they are distributed between the two sockets.

```
MPI: MPI_DSM_DISTRIBUTE enabled
grank   lrank   pinning  node name      cpuid
    0       0   yes      r212i0n10          0
    1       1   yes      r212i0n10          1
    2       2   yes      r212i0n10          2
    3       3   yes      r212i0n10          3
    4       0   yes      r212i0n11          0
    5       1   yes      r212i0n11          1
    6       2   yes      r212i0n11          2
    7       3   yes      r212i0n11          3
```

The second problem is that, as demonstrated with the **ps** command below, the OpenMP threads are also placed on the same core where the associated MPI process is running:

```
ps -C a.out -L -opsr,comm,time,pid,ppid,lwp
PSR COMMAND         TIME   PID  PPID   LWP
  0 a.out       00:00:02  4098  4092  4098
  0 a.out       00:00:02  4098  4092  4108
  0 a.out       00:00:02  4098  4092  4110
  1 a.out       00:00:03  4099  4092  4099
  1 a.out       00:00:03  4099  4092  4106
  2 a.out       00:00:03  4100  4092  4100
  2 a.out       00:00:03  4100  4092  4109
  3 a.out       00:00:03  4101  4092  4101
  3 a.out       00:00:03  4101  4092  4107
```

Sample PBS script demonstrating proper placement:

```
#PBS -l select=2:ncpus=12:mpiprocs=4:model=wes

module load mpi-sgi/mpt.2.04.10789
module load comp-intel/11.1.072

setenv MPI_DSM_VERBOSE
setenv OMP_NUM_THREADS 2
setenv KMP_AFFINITY disabled

cd $PBS_O_WORKDIR

#the following two lines will result in identical placement

mpiexec -np 8 omplace -nt 2 -c 0,1,3,4,6,7,9,10 -vv ./a.out
#mpiexec -np 8 omplace -nt 2 -c 0-10:bs=2+st=3 -vv  ./a.out
```

Shown in the PBS **`stderr`** file, the 4 MPI processes on each node are properly distributed on the two sockets with processes 0 and 1 on CPUs 0 and 3 (first socket) and processes 2 and 3 on CPUs 6 and 9 (second socket).

```
MPI: DSM information
MPI: using dplace
grank   lrank   pinning  node name      cpuid
    0       0   yes      r212i0n10          0
    1       1   yes      r212i0n10          3
    2       2   yes      r212i0n10          6
    3       3   yes      r212i0n10          9
    4       0   yes      r212i0n11          0
    5       1   yes      r212i0n11          3
    6       2   yes      r212i0n11          6
    7       3   yes      r212i0n11          9
```

In the PBS **`stout`** file, it shows the placement of the two OpenMP threads for each MPI process:

```
omplace: This is an SGI MPI program.
omplace: placement file /tmp/omplace.file.6454
    fork skip=0  exact cpu=0-10:3
    thread oncpu=0 cpu=1 noplace=1  exact
    thread oncpu=3 cpu=4 noplace=1  exact
    thread oncpu=6 cpu=7 noplace=1  exact
    thread oncpu=9 cpu=10 noplace=1  exact
omplace: This is an SGI MPI program.
omplace: placement file /tmp/omplace.file.22771
    fork skip=0  exact cpu=0-10:3
    thread oncpu=0 cpu=1 noplace=1  exact
    thread oncpu=3 cpu=4 noplace=1  exact
    thread oncpu=6 cpu=7 noplace=1  exact
    thread oncpu=9 cpu=10 noplace=1  exact
```

To get a better picture of how the OpenMP threads are placed, using the following **`ps`** command:

```
ps -C a.out -L -opsr,comm,time,pid,ppid,lwp
PSR COMMAND         TIME   PID  PPID   LWP
  0 a.out       00:00:06  4436  4435  4436
  1 a.out       00:00:03  4436  4435  4447
  1 a.out       00:00:03  4436  4435  4448
  3 a.out       00:00:06  4437  4435  4437
  4 a.out       00:00:05  4437  4435  4446
  6 a.out       00:00:06  4438  4435  4438
  7 a.out       00:00:05  4438  4435  4444
  9 a.out       00:00:06  4439  4435  4439
 10 a.out       00:00:05  4439  4435  4445
```

Article ID: 287
Last updated: 08 Aug, 2012
Computing at NAS -> Best Practices -> Process Pinning -> Using SGI omplace for Pinning
http://www.nas.nasa.gov/hecc/support/kb/entry/287/?ajax=1